САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ – ПРОЦЕССОВ УПРАВЛЕНИЯ

ОТЧЕТ

по лабораторной работе №10

по дисциплине «Алгоритмы и структуры данных»

на тему «Муравьиный алгоритм»

Выполнила
студентка 2 курса
группы 21-Б15.ПУ
Павлова Ксения Андреевна

Преподаватель
Дик Александр Геннадьевич

Санкт-Петербург
2023

СОДЕРЖАНИЕ

Цель: исследование особенностей решения задачи о коммивояжере с помощью муравьиного алгоритма и сравнение алгоритма с методом ближайшего соседа и алгоритмом имитации отжига.

Задачи:

1. Формализовать задачу о коммивояжере с помощью муравьиного алгоритма.
2. Подготовить контрольный пример, используя взвешенный граф.
3. Найти кратчайший гамильтонов цикл.
4. Сравнить решение задачи о коммивояжере с помощью алгоритма ближайшего соседа (с результатами лабораторной работы №8 и №9).
5. Написать пользовательский интерфейс для удобства пользования алгоритмом.

Ход работы.

Краткое описание муравьиного алгоритма.

Муравьиные алгоритмы представляют собой вероятностную жадную эвристику, где вероятности устанавливаются, исходя из информации о качестве решения, полученной из предыдущих решений.

Идея муравьиного алгоритма – моделирование поведения муравьёв, связанного с их способностью быстро находить кратчайший путь от муравейника к источнику пищи и адаптироваться к изменяющимся условиям, находя новый кратчайший путь. При своём движении муравей метит путь феромоном, и эта информация используется другими муравьями для выбора пути. Это элементарное правило поведения и определяет способность муравьёв находить новый путь, если старый оказывается недоступным.

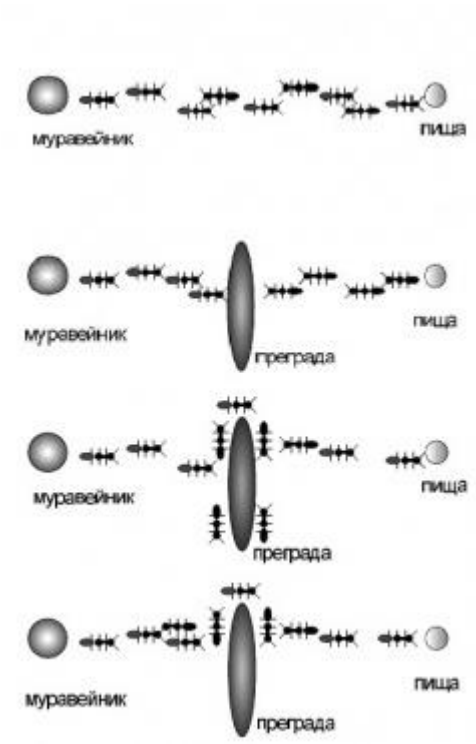

Рис. 1. Поведение колонии

Рассмотрим случай, показанный на рисунке 1, когда на оптимальном доселе пути возникает преграда. В этом случае необходимо определение нового оптимального пути. Дойдя до преграды, муравьи с равной вероятностью будут обходить её справа и слева. То же самое будет происходить и на обратной стороне преграды. Однако, те муравьи, которые

случайно выберут кратчайший путь, будут быстрее его проходить, и за несколько передвижений он будет более обогащён феромоном. Поскольку движение муравьёв определяется концентрацией феромона, то следующие будут предпочитать именно этот путь, продолжая обогащать его феромоном до тех пор, пока этот путь по какой-либо причине не станет недоступен.

Очевидная положительная обратная связь быстро приведёт к тому, что кратчайший путь станет единственным маршрутом движения большинства муравьёв. Моделирование испарения феромона – отрицательной обратной связи – гарантирует нам, что найденное локально оптимальное решение не будет единственным – муравьи будут искать и другие пути. Если мы моделируем процесс такого поведения на некотором графе, рёбра которого представляют собой возможные пути перемещения муравьёв, в течение определённого времени, то наиболее обогащённый феромоном путь по рёбрам этого графа и будет являться решением задачи, полученным с помощью муравьиного алгоритма.

Описание схемы пошагового выполнения алгоритма.

Любой муравьиный алгоритм, независимо от модификаций, представим в следующем виде

Пока (условия выхода не выполнены):

1. Создаём муравьёв

Стартовая точка, куда помещается муравей, зависит от ограничений, накладываемых условиями задачи. Потому что для каждой задачи способ размещение муравьёв является определяющим. Либо все они помещаются в одну точку, либо в разные с повторениями, либо без повторений.

На этом же этапе задаётся начальный уровень феромона. Он инициализируется небольшим положительным числом для того, чтобы на начальном шаге вероятности перехода в следующую вершину не были нулевыми.

2. Ищем решения

Вероятность перехода из вершины i в вершину j определяется по следующей формуле

$$P_{ij,k}(t) = \frac{[\tau_{ij}(t)]^{\alpha} \cdot [\eta_{ij}]^{\beta}}{\sum_{l \in J_{i,k}} [\tau_{il}(t)]^{\alpha} \cdot [\eta_{il}]^{\beta}},$$

где $\tau_{ij}(t)$ – уровень феромона, η_{ij} – эвристическое расстояние, α, β – константные параметры. Необходим компромисс между этими величинами (чтоб смягчать жадность алгоритма и не застревать в локальных минимумах), который находится экспериментально.

3. Обновляем феромон

Уровень феромона обновляется в соответствии с приведённой формулой

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \sum_{k \in \{used(i,j)\}} \frac{Q}{L_k(t)},$$

где ρ – интенсивность испарения, $L_k(t)$ – цена текущего решения для k-ого муравья, а Q – параметр, имеющий значение порядка цены оптимального решения, то есть $Q/L_k(t)$ - феромон, откладываемый k-ым муравьём, использующим ребро (i,j).

4. Дополнительные действия (опционально)

Обычно здесь используется алгоритм локального поиска, однако он может также появиться и после поиска всех решений.

Формализация задачи.

Для решения поставленной задачи был выбран язык программирования Python. Визуализация выполнялась с помощью библиотеки TKinter.

1. Создаём муравьёв.

С помощью цикла создаётся список муравьев. Количество муравьев определяется параметром «num_ants». По умолчанию количество муравьёв равно 10. Каждому муравью случайным образом назначается стартовая

вершина. Создается пустое множество «nodes», в которое добавляются все вершины графа. Для каждого ребра в графе вычисляется его длина и добавляется в словарь «distances» с ключом в виде кортежа из двух вершин, а также добавляется в словарь «pheromones» с начальным значением феромона равным 1.

2. Ищем решения.

Вероятность зависит от расстояния между вершинами и количества феромонов на ребре. Выбор следующей вершины осуществляется с помощью функции «prob», которая возвращает список вероятностей для каждого ребра.

Параметры α и β для вычисления вероятности задаются в пользовательском интерфейсе. Их значения по умолчанию равны 1 и 5 соответственно.

3. Обновляем феромон

Испарение феромона происходит на каждой итерации алгоритма. Для этого на каждой итерации значение феромона на всех ребрах уменьшается на заданный коэффициент испарения «evaporation_rate». Коэффициент испарения задаётся в интерфейсе пользователя. По умолчанию он равен 0,1. Это делается для того, чтобы предотвратить слишком быстрое накопление феромона на некоторых ребрах и обеспечить более равномерное распределение феромона по графу. Также это позволяет избежать застревания в локальных минимумах.

Обновление феромона происходит в конце каждой итерации алгоритма. Для каждого муравья находится его путь и вычисляется общее расстояние, которое он прошел. Затем для каждого ребра на его пути увеличивается значение феромона на величину, обратную этому расстоянию, умноженной на некоторую константу «pheromone_constant». Эта константа – мощность феромона – задаётся в интерфейсе пользователя. Значение по умолчанию равно 2. Это делается для того, чтобы ребра, по которым прошли успешные

муравьи, получали больше феромона и становились более привлекательными для следующих муравьев.

4. Алгоритм останавливается после прохождения заданного количества итераций. Количество итераций по умолчанию 45.

Листинг программы.

Муравьиный алгоритм.

```
#Блок подключения необходимых библиотек
import random

#Запуск муравьиного алгоритма
def ant_alg(mass):
    #Инициализация
    canvas = mass[0]
    canvas.best_edges_delete()
    canvas.edges_mass =
    canvas.table.check_data(canvas.edges_mass)
    edges = canvas.edges_mass
    num_ants = int(mass[1].get())
    iterations = int(mass[2].get())
    evaporation_rate = float(mass[3].get())
    alpha = float(mass[4].get())
    beta = float(mass[5].get())
    pheromone_constant = float(mass[6].get())
    ant_colony = AntColony(num_ants, edges, evaporation_rate,
    pheromone_constant, iterations,
                           alpha, beta)
    ant_colony.run()
    canvas.best_way = ant_colony.best_path
    canvas.best_weight = ant_colony.best_distance
    canvas.all_edges_hidden(True)
    canvas.draw_best_way()
    return canvas

#Задание муравья
class Ant:
    def __init__(self, id, start_node):
        self.id = id
        self.path = [start_node]
        self.visited_nodes = set()
        self.visited_nodes.add(start_node)
        self.total_distance = 0
    #Функция движения к узлу
    def move_to_node(self, node, distance):
        self.path.append(node)
        self.visited_nodes.add(node)
        self.total_distance += distance
```

```
#Задание колонии муравьёв и её поведения
class AntColony:
    def __init__(self, num_ants, edges, evaporation_rate,
pheromone_constant, iterations,alpha,beta):
        self.num_ants = num_ants
        self.edges = edges
        self.evaporation_rate = evaporation_rate #испарение
феромона
        self.pheromone_constant = pheromone_constant #отложение
феромона
        self.iterations = iterations
        self.alpha = alpha
        self.beta = beta
        self.nodes = set()
        self.distances = {}
        self.pheromones = {}
        self.best_path = None
        self.best_distance = float('inf')

    #Поиск решений
        for edge in edges:
            node1, node2, distance = edge
            self.nodes.add(node1)
            self.nodes.add(node2)
            self.distances[(node1, node2)] = distance
            self.distances[(node2, node1)] = distance
            self.pheromones[(node1, node2)] = 1
            self.pheromones[(node2, node1)] = 1

    #Функция вероятности
    def prob(self,edges,distance,pheromones):
        sum = 0
        mass = []
        for i in range(len(edges)):
            d = distance[i]
            t = pheromones[i]
            h = d**self.alpha*(1/d)**self.beta
            mass.append(h)
            sum += h
        for i in range(len(mass)):
            mass[i] = mass[i] / sum
        return mass

    #Функция движения колонии
    def run(self):
        for i in range(self.iterations):
            ants = [Ant(j, random.choice(list(self.nodes))) for
j in range(self.num_ants)]
            for ant in ants:
                while len(ant.visited_nodes) < len(self.nodes):
                    current_node = ant.path[-1]
                    candidate_nodes = [node for node in
```

```
self.nodes if node not in ant.visited_nodes]
                    if len(candidate_nodes) == 0:
                        break
                    candidate_edges = [(current_node, node) for
node in candidate_nodes]
                    candidate_distances = [self.distances[edge]
for edge in candidate_edges]
                    candidate_pheromones =
[self.pheromones[edge] for edge in candidate_edges]
                    candidate_probabilities =
self.prob(candidate_edges,candidate_distances,candidate_pheromon
es)
                    next_node = random.choices(candidate_nodes,
candidate_probabilities)[0]
                    distance = self.distances[(current_node,
next_node)]
                    ant.move_to_node(next_node, distance)
                ant.move_to_node(ant.path[0],
self.distances[(ant.path[-1], ant.path[0])])
                if ant.total_distance < self.best_distance:
                    self.best_path = ant.path
                    self.best_distance = ant.total_distance
            for edge in self.pheromones:
                self.pheromones[edge] *= (1 -
self.evaporation_rate)
            for ant in ants:
                for i in range(len(ant.path) - 1):
                    edge = (ant.path[i], ant.path[i+1])
                    self.pheromones[edge] +=
self.pheromone_constant / ant.total_distance
```

Визуализация.

```
#Блок подключения необходимых библиотек
import tkinter as tk
from tkinter import ttk
from tkinter import simpledialog

class myWindow:
    def __init__(self, root, color):
        self.root = root
        self.color = color
        #Создание главных фреймов
        self.left_frame = tk.Frame(self.root, height=250,
width=500)
        self.right_frame = tk.Frame(self.root, height=250,
width=500)
        self.left_frame.grid(row=0, column=0, sticky="nsew")
        self.right_frame.grid(row=0, column=1, sticky="nsew")
        #Задание пропорций растяжения для колонок
        self.root.columnconfigure(0, weight=1)
        self.root.columnconfigure(1, weight=1)
        #Задание обработчика изменения размера окна
        self.root.bind("<Configure>", self.on_resize)
```

```
        #Создание фреймов в левом фрейме
        self.top_left_frame = tk.LabelFrame(self.left_frame,
text='Input area', bg=self.color, height=125, width=500)
        self.middle_left_frame = tk.LabelFrame(self.left_frame,
text='Table area', bg=self.color, height=125, width=500)
        self.bottom_left_frame = tk.LabelFrame(self.left_frame,
text='Result area', bg=self.color, height=125, width=500)
        self.top_left_frame.pack(side="top", fill="both",
expand=True)
        self.middle_left_frame.pack(side="top", fill="both",
expand=True)
        self.bottom_left_frame.pack(side="bottom", fill="both",
expand=True)
        #Создание фреймов в правом фрейме
        self.top_right_frame = tk.LabelFrame(self.right_frame,
text='Canvas area', bg=self.color, height=125, width=500)
        self.top_right_frame.pack(side="top", fill="both",
expand=True)

    #Функция запуска
    def on_resize(self, event):
        #Получение текущей ширины окна
        width = self.root.winfo_width()
        #Вычисление новой ширины главных фреймов
        left_width = int(width / 3)
        right_width = width - left_width
        #Изменение ширины колонок
        self.root.columnconfigure(0, minsize=left_width)
        self.root.columnconfigure(1, minsize=right_width)

#Общее описание кнопок
class Button:
    def __init__(self, root,text,command,arg):
        self.arg = arg
        self.command = command
        self.root = root
        self.text = text
        button = tk.Button(self.root, text=self.text,
command=lambda: self.command(self.arg))
        button.pack()

#Описание полей с вводом значений
class labeledSpinbox():
    def __init__(self, root,label_text, spinbox_from,
spinbox_to, spinbox_default,spinbox_step):
        self.root = root
        self.label_text = label_text
        self.spinbox_from = spinbox_from
        self.spinbox_to = spinbox_to
        self.spinbox_default = spinbox_default
        self.spinbox_step = spinbox_step
        #Создание виджета Label
```

```
        self.widget_frame = tk.Frame(self.root)
        self.widget_frame.pack(side=tk.TOP)
        self.label = tk.Label(self.widget_frame,
text=self.label_text,font=("Arial", 10))
        self.label.pack(side=tk.LEFT)
        # Создание виджета Spinbox
        self.spinbox = tk.Spinbox(self.widget_frame,
from_=self.spinbox_from, to=self.spinbox_to, width=10,
increment=self.spinbox_step,font=("Arial", 10))
        self.spinbox.delete(0, tk.END)  #Удаление стандартного
значения
        self.spinbox.insert(0, spinbox_default)  #Установка
стандартного значения
        self.spinbox.pack(side=tk.LEFT)

    def get(self):
        return self.spinbox.get()
#Создание холстов и рисование графов
class canvas:
    def __init__(self, root,table,result):
        self.root = root
        self.table = table
        self.result = result
        self.canvas = tk.Canvas(self.root, width=400,
height=400, bg='white')
        self.canvas.pack(fill="both", expand=True)
        self.circle_count = 0
        self.circle_mass = []
        self.edges_mass = []
        self.best_way = []
        self.best_weight=0
        self.counter = 0
        #Привязка функции рисования к событию нажатия мыши
        self.canvas.bind("<Button-1>", self.draw_circle)
        self.canvas.bind("<Button-3>", self.circle_delete)
        self.canvas.bind("<Button-2>", self.show_data)

    #Функция рисования круга на холсте
    def show_data(self,event):
        print(self.edges_mass)
        print(self.circle_mass)
    def draw_circle(self, event):
        self.best_edges_delete()
        self.all_edges_hidden(False)
        x, y = event.x, event.y
        r = 20
        overlapping_circles = self.canvas.find_enclosed(x - r-
        25, y - r-25, x + r+25, y + r+25)  #поиск
перекрывающихся кругов
        if not overlapping_circles:  #если нет пересечений
            #Добавление новой вершины
            self.circle_mass.append([self.circle_count,
```

```
            [int(x),int(y)]])
            self.canvas.create_oval(x - r, y - r, x + r, y + r,
            fill='red',tags='oval')
            self.canvas.create_text(x, y,
            text=str(self.circle_count), font=('Arial', 12,
            'bold'),tags='text')
            self.edges_mass =
            self.table.check_data(self.edges_mass)
            self.draw_edges(self.circle_count)
            self.circle_count += 1
            self.table.update_data(self.edges_mass)
        self.canvas.tag_raise('oval')
        self.canvas.tag_raise('text')

    #Рисование рёбер
    def draw_edges(self, id):
        l = len(self.circle_mass)
        for i in range(l-1):
            x1, y1 = self.circle_mass[l-1][1]
            x2, y2 = self.circle_mass[i][1]
            id1 = self.circle_mass[l-1][0]
            id2 = self.circle_mass[i][0]
            self.canvas.create_line(x1, y1, x2, y2, width=1,
            fill="black",tags='lines')
            weight = ((x2 - x1)**2 + (y2 - y1)**2)**0.5
            self.edges_mass.append([id1,id2,int(weight)])

    #Удаление вершины
    def circle_delete(self, event):
        self.edges_mass = self.table.check_data(self.edges_mass)
        x = event.x
        y = event.y
        r = 20
        objects = self.canvas.find_overlapping(x-r,y-r,x+r,y+r)
        for obj in objects:
            if self.canvas.type(obj) == 'oval':
                coords = self.canvas.coords(obj)
                circle_x = (coords[0] + coords[2]) // 2
                circle_y = (coords[1] + coords[3]) // 2
                for oval in self.circle_mass:
                    if oval[1][0] == circle_x and oval[1][1] ==
                    circle_y:
                        self.delId = oval[0]
                        self.circle_mass.remove(oval)
                        self.edges_delete()
            self.canvas.delete(obj)
        self.table.update_data(self.edges_mass)

    #удаление ребра
    def edges_delete(self):
        l = len(self.edges_mass)
        self.edges_mass = [inner_lst for inner_lst in
```

```
        self.edges_mass if self.delId not in inner_lst]

    #скрытие и показ рёбер
    def all_edges_hidden(self,Bool):
        items = self.canvas.find_withtag("lines")
        for item in items:
            if Bool:
                self.canvas.itemconfig(item, state='hidden')
            else:
                self.canvas.itemconfig(item, state='normal')

    #Удаление рёбер найденного пути
    def best_edges_delete(self):
        items = self.canvas.find_withtag("bestlines")
        for item in items:
          self.canvas.delete(item)

    #Отрисовка найденного пути
    def draw_best_way(self):
        self.result.update_data_res(self.best_way,
        self.best_weight,self.counter)
        coordinates = []
        for i in self.best_way:
            for j in self.circle_mass:
                if j[0] == i:
                    coordinates.append(j[1])
        coordinates.append(coordinates[0])
        for i in range(len(coordinates)-1):
            self.canvas.create_line(coordinates[i][0],
coordinates[i][1], coordinates[i+1][0], coordinates[i+1][1],
width=2, fill="red",tags='bestlines')
        self.canvas.tag_raise('text')

    #Очистка холста
    def clear_canvas(self,bool):
        self.canvas.delete("all")
        self.edges_mass = []
        self.circle_mass = []
        self.best_way = []
        self.best_weight = 0
        self.circle_count=0
        self.counter = 0
        self.table.update_data(self.edges_mass)
        self.result.update_data_res(self.best_way,
        self.best_weight,self.counter)

#Задание таблицы смежности
class myTable:
    def __init__(self,root,headers):
        self.root = root
        self.headers = headers
        self.changed_mass = []
```

```
        self.changed_val = []
        self.table = ttk.Treeview(self.root,
        columns=self.headers, show='headings')
        for header in self.headers:
            self.table.heading(header, text=header.title())
        for col in self.table["columns"]:
            self.table.column(col, width=100)
        self.table.pack(side=tk.TOP, fill=tk.X, expand=1)
        self.table.bind("<Double-1>", self.update_cell)

    #Обновление данных
    def update_data(self,mass):
        self.table.delete(*self.table.get_children())
        for row in mass:
            self.table.insert("", "end", values=row)
    #Проверка данных
    def check_data(self,list1):
        if len(self.changed_mass):
            for i in range(len(list1)):
                if list1[i] in self.changed_mass:
                    list1[i][2] = self.changed_val.pop(0)
            self.changed_mass = []
            self.changed_val = []
        return list1
    #Обновление значение ячеек
    def update_cell(self,event):
        #Получение ссылки на таблицу и на выбранную строку
        self.table = event.widget
        self.item = self.table.selection()[0]
        #Получение индекса выбранной колонки и имени выбранного
столбца
        self.column = self.table.identify_column(event.x)
        self.column_name =
self.table.heading(self.column)['text']
        #Получение старого значения ячейки и запрос у
пользователя нового значения
        self.old_value = self.table.set(self.item, self.column)
        self.id1 = self.table.set(self.item, '#1')
        self.id2 = self.table.set(self.item, '#2')
        self.new_value = simpledialog.askstring('Изменение
значения', f'Введите новое значение для
{self.column_name}:',initialvalue=self.old_value)
        #Обновление значения ячейки, если пользователь ввёл
новое значение
        if self.new_value:
            self.changed_mass.append([int(self.id1),
            int(self.id2),int(self.old_value)])
            self.changed_val.append(int(self.new_value))
            self.table.set(self.item, self.column,
            self.new_value)
```

```
#Задание блока вывода результатов
class myResult:
    def __init__(self,root):
        self.root = root
        self.visited = []
        self.best_weight = 0
        self.counter = 0
        self.label = tk.Label(self.root, text=f"Путь:
        {self.visited}")
        self.label2 = tk.Label(self.root, text=f"Общий вес пути:
        {self.best_weight}")
        self.label3 = tk.Label(self.root, text=f"Количество
        поколений: {self.counter}")
        self.label.pack()
        self.label2.pack()
    #Обновление результатов
    def update_data_res(self,vis,wg,counter):
        self.visited = vis
        self.best_weight = wg
        self.counter = counter
        self.label.config(text=f"Путь: {self.visited}")
        self.label2.config(text=f"Общий вес пути:
        {self.best_weight}")
        if self.counter:
            self.label3.config(text=f"Количество поколений:
            {self.counter}")
            self.label3.pack()
```

Подключение визуализации и основного алгоритма.

```
#Блок подключения необходимых библиотек
import tkinter as tk
from tkinter import ttk, messagebox
import gui
from alg import nearest_neighbor

#Функция закрытия программы
def on_closing():
    if messagebox.askokcancel("Выход", "Вы действительно хотите
выйти?"): root.destroy()

#Создание окна с вкладками и фреймов
root = tk.Tk()
root.geometry("1000x500")
root.protocol("WM_DELETE_WINDOW", on_closing)

notebook = ttk.Notebook(root)
notebook.pack(fill='both', expand=True)

tab1 = tk.Frame(notebook)
tab2 = tk.Frame(notebook)
tab3 = tk.Frame(notebook)
```

```
notebook.add(tab1, text='Метод ближайшего соседа')

#Запуск муравьиного алгоритма
ant_window = gui.myWindow(tab3,'#e8caca')
ant_table = gui.myTable(ant_window.middle_left_frame,['Вершина
1', 'Вершина 2', 'Вес'])
ant_result = gui.myResult(ant_window.bottom_left_frame)
ant_canvas =
gui.canvas(ant_window.top_right_frame,ant_table,ant_result)
ant_value =
gui.labeledSpinbox(ant_window.top_left_frame,'Количество
муравьев',1,1000,10,5)
ant_iter =
gui.labeledSpinbox(ant_window.top_left_frame,'Количество
поколений',1,1000,45,5)
ant_alpha = gui.labeledSpinbox(ant_window.top_left_frame,'Альфа
значение',1,10,1,1)
ant_beta = gui.labeledSpinbox(ant_window.top_left_frame,'Бета
значение',1,10,5,1)
ant_decay =
gui.labeledSpinbox(ant_window.top_left_frame,'Скорость распада
феромона',0,10,0.1,0.1)
ant_pherconst =
gui.labeledSpinbox(ant_window.top_left_frame,'Мощность
феромона',0,10,2,1)
ant_startButton = gui.Button(ant_window.top_left_frame,'Начать
алгоритм',ant_alg.ant_alg,[ant_canvas,ant_value,ant_iter,ant_dec
ay,ant_alpha,ant_beta,ant_pherconst])
ant_clearButton = gui.Button(ant_window.top_left_frame,'Стереть
данные',ant_canvas.clear_canvas,True)


root.mainloop()
```

Спецификация программы.

Пользовательский интерфейс (рис. 1)

Для создания графа используется правый фрейм. При нажатии ЛКМ создаётся вершина, которая соединяется со всеми остальными вершинами. Каждому ребру задаётся вес в соответствии с расстоянием до других вершин. Чтобы удалить вершину или ребро нужно нажать на соответствующий элемент ПКМ. Чтобы запустить алгоритм надо нажать кнопку «Начать алгоритм» в левой верхней части окна. Под ней располагается кнопка «Стереть данные», позволяющая очистить область рисования. Также в середине левого фрейма находится таблица со списком рёбер и весами. У каждого ребра можно вручную изменить вес, нажав на него в таблице дважды ЛКМ. В нижней части

левого фрейма находится область вывода результата. При нажатии на знак крестика в правом верхнем углу осуществляется выход из программы, который нужно подтвердить. Если не ввести граф, то алгоритм выдаст ошибку «ValueError: Sample larger than population or is negative».

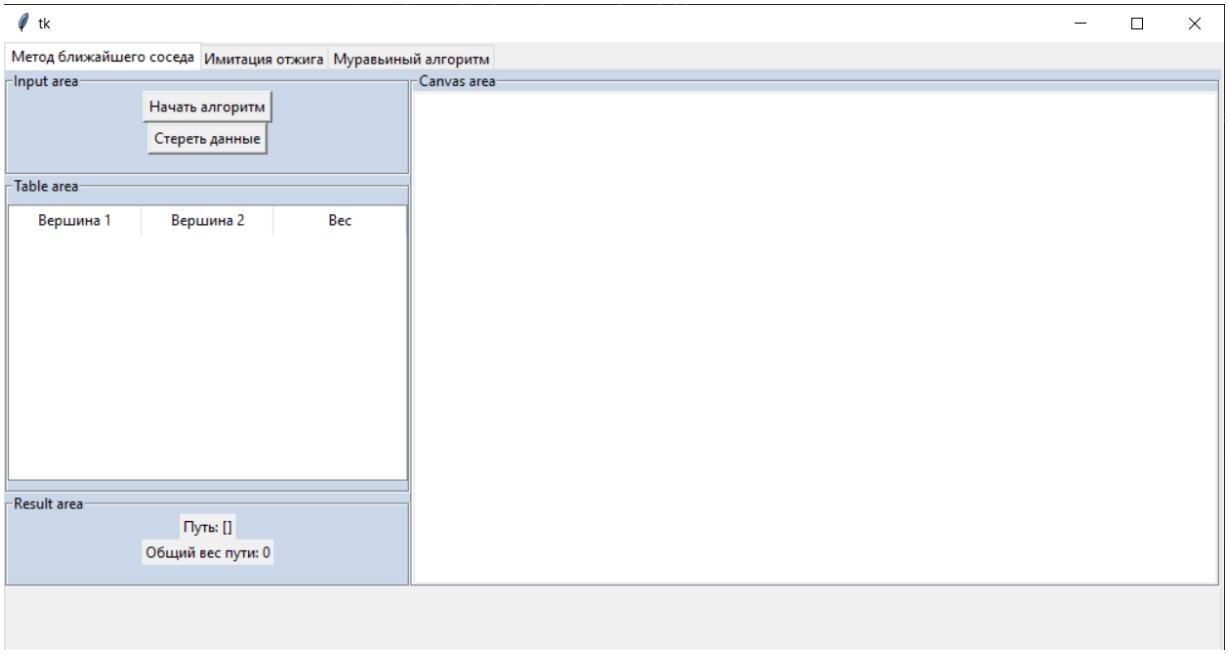

Рис. 2. Пользовательский интерфейс

Для создания графа используется правый фрейм. При нажатии ЛКМ создаётся вершина, которая соединяется со всеми остальными вершинами. Каждому ребру задаётся вес в соответствии с расстоянием до других вершин. Чтобы удалить вершину или ребро нужно нажать на соответствующий элемент ПКМ. Чтобы запустить алгоритм надо нажать кнопку «Начать алгоритм» в левой верхней части окна. Под ней располагается кнопка «Стереть данные», позволяющая очистить область рисования. Также в середине левого фрейма находится таблица со списком рёбер и весами. У каждого ребра можно вручную изменить вес, нажав на него в таблице дважды ЛКМ. В нижней части левого фрейма находится область вывода результата. При нажатии на знак крестика в правом верхнем углу осуществляется выход из программы, который нужно подтвердить. Если не ввести граф, то алгоритм выдаст ошибку « IndexError: list index out of range».

Контрольный пример.

Тестирование программы производилось на графе, представленном на рис. 3 с случайно заданными весами рёбер и параметрами по умолчанию.

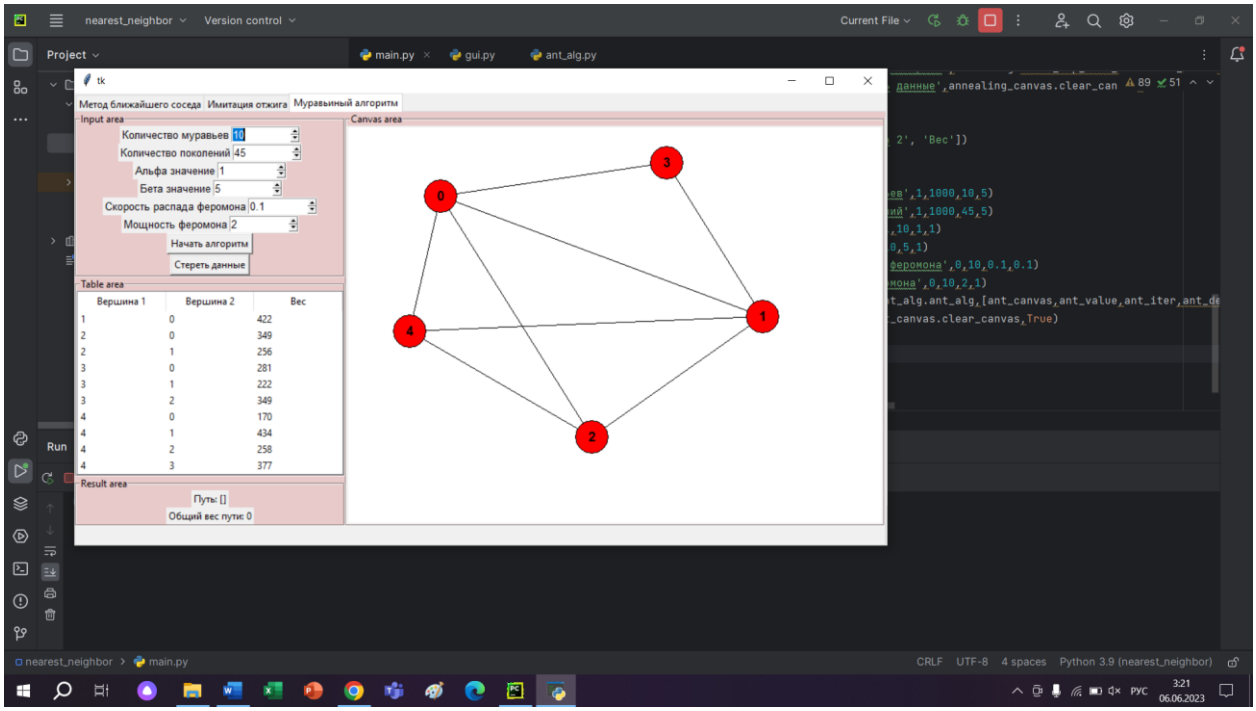

Рис. 3. Начало работы алгоритма

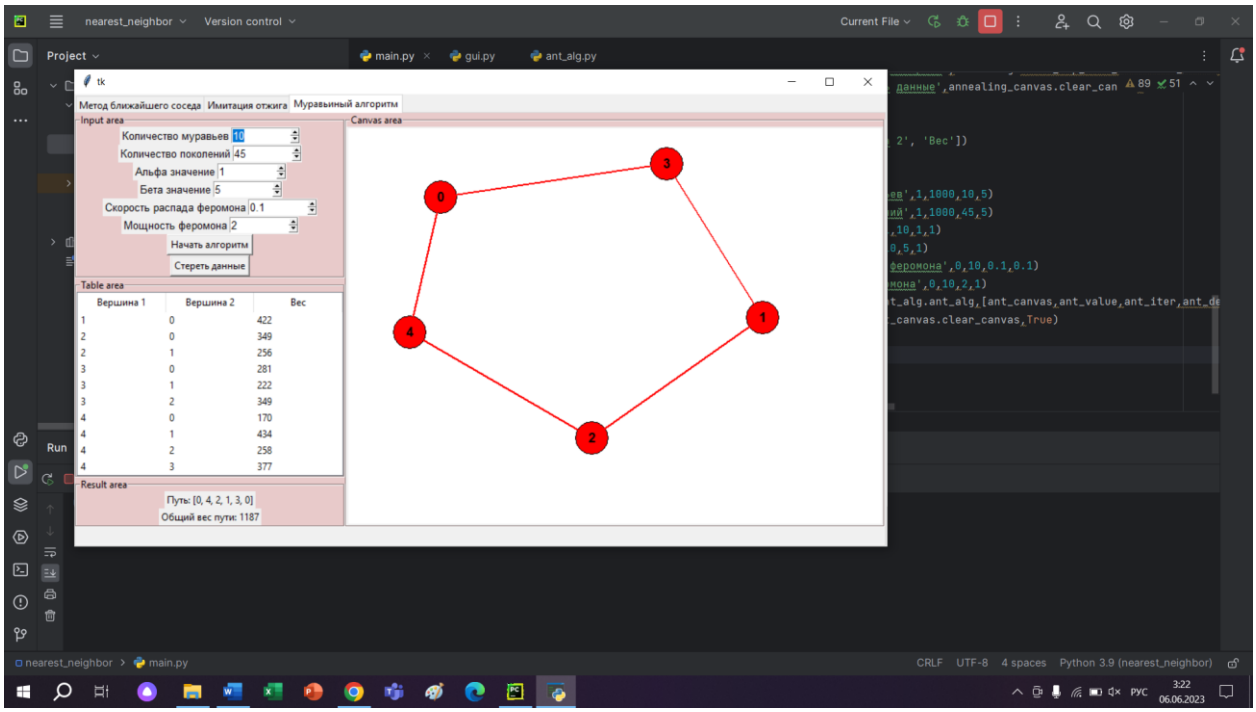

Рис. 4. Окончание работы алгоритма

Таким образом, найден кратчайший гамильтонов цикл, представленный на рис. 4, путь цикла по вершинам: 0, 4, 2, 1, 3, 0.

Анализ результатов работы алгоритма.

Анализ результатов работы алгоритма проводился с помощью тестового графа, представленного на рис. 5.

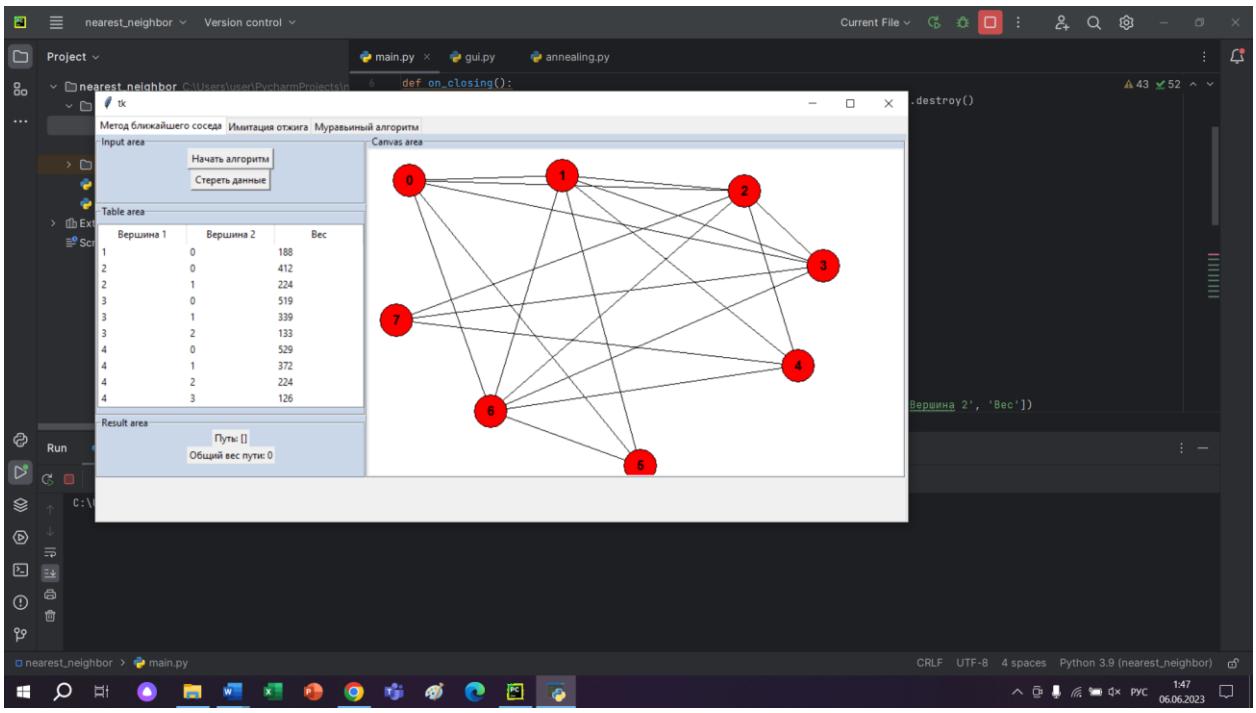

Рис. 5. Тестовый граф

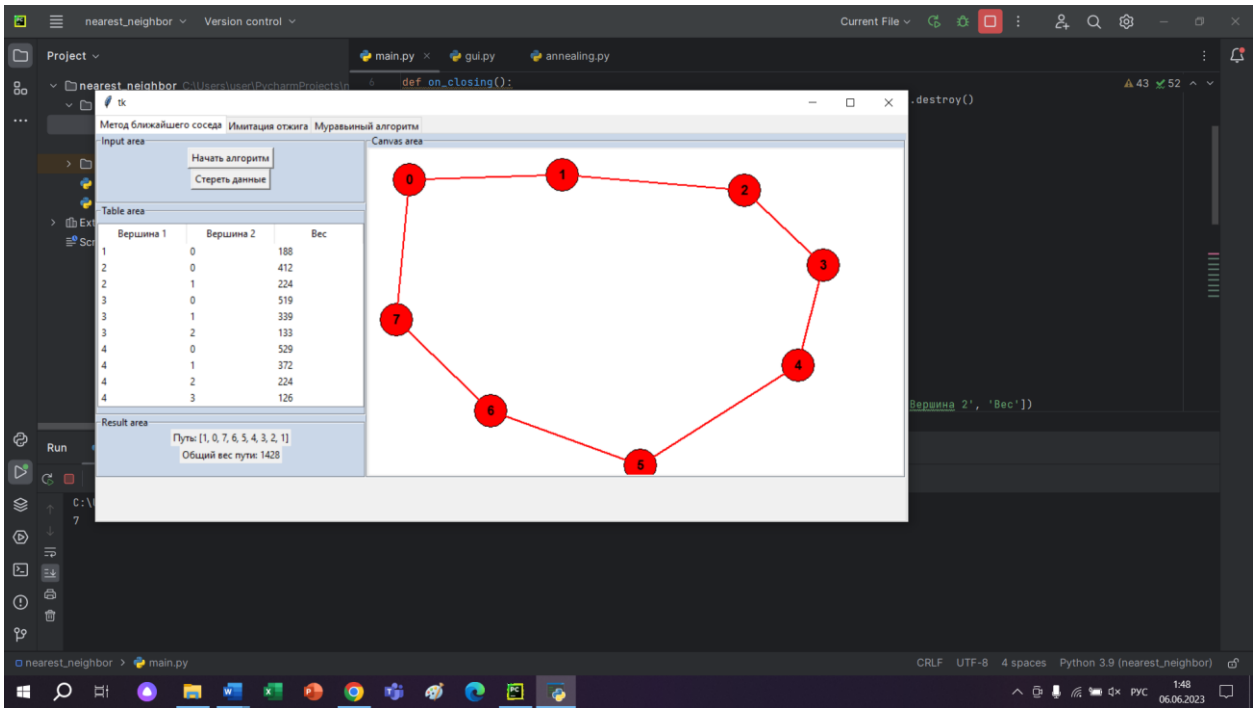

Рис. 6. Результат работы метода ближайшего соседа.

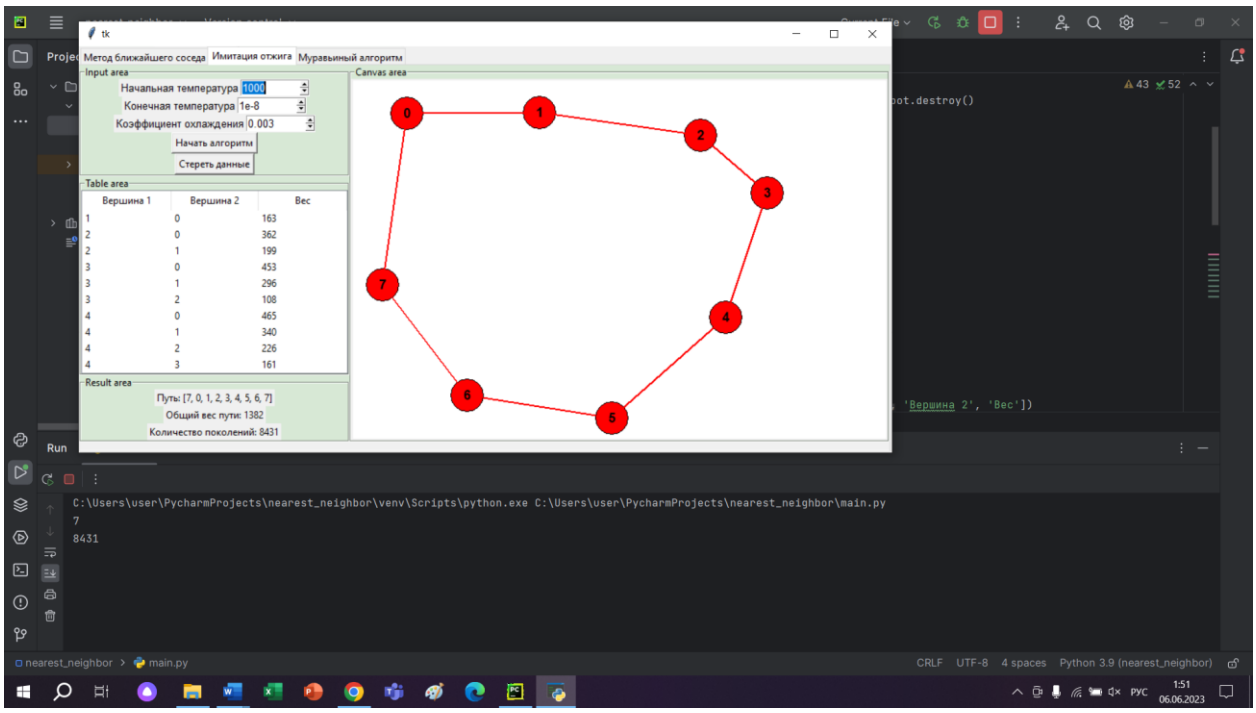

Рис. 7. Результат работы алгоритма имитации отжига

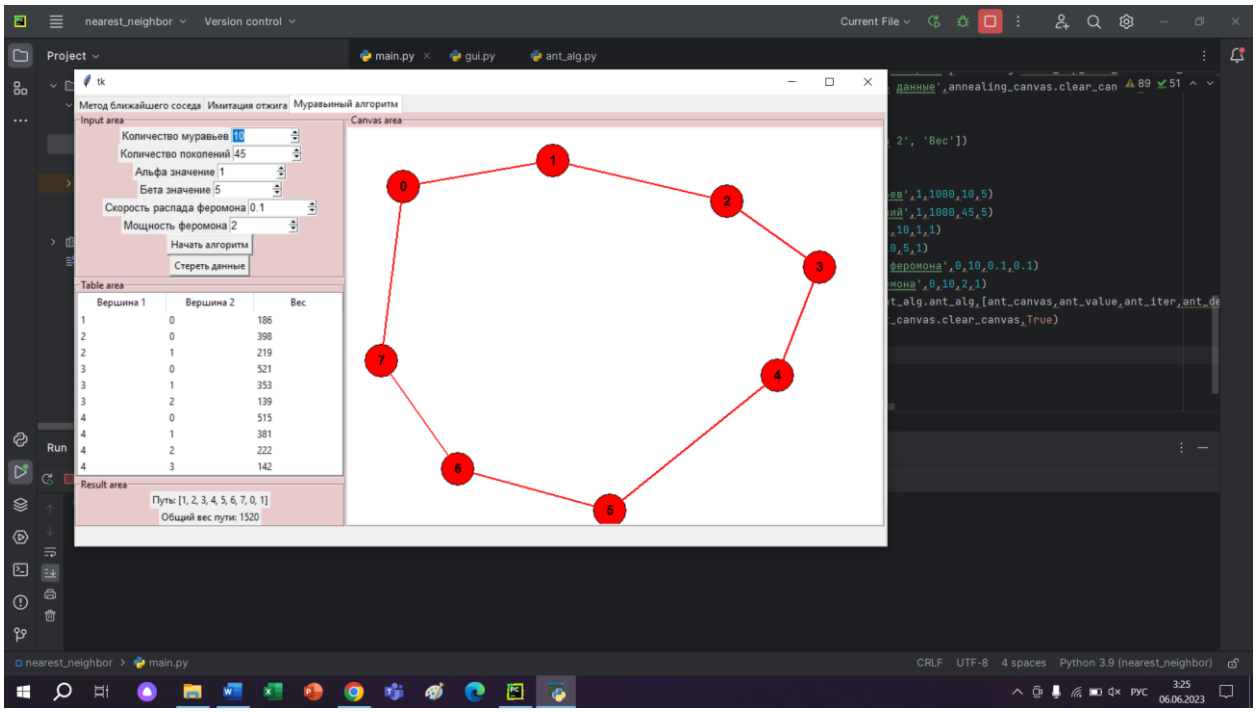

Рис. 8. Результат работы муравьиного алгоритма

Анализ результатов работы алгоритма показал, что муравьиный алгоритм способен справляться со сложными графами и искать глобальный оптимум. Муравьиный алгоритм даёт более точное решение за счёт исследования более широкого пространства решений и является более эффективным для задач, требующих минимального времени выполнения.

Заключение.

В ходе работы изучен и реализован муравьиный алгоритм, а также проведён анализ результатов работы алгоритма и сравнение алгоритма с методом ближайшего соседа и алгоритмом имитации отжига. Из полученных данных сделаны выводы:

- Метод ближайшего соседа является самым простым в реализации и имеет линейную сложность, однако не всегда находит оптимальное решение.
- За счёт вероятностного подхода и исследования более широкого пространства решений муравьиный алгоритм более точное решение, чем алгоритм имитации отжига, а значит и чем метод ближайшего соседа.
- Муравьиный алгоритм более эффективен по времени, чем алгоритм имитации отжига, но менее эффективен, чем метод ближайшего соседа.
- Муравьиный алгоритм имеет более широкий набор параметров для настройки, чем алгоритм имитации отжига, что может осложнить выбор оптимальных параметров.

Список литературы.

1. Introducing Python. Modern computing in simple packages / Bill Lobanovic.
2. http://www.machinelearning.ru/
3. Ссылка на код https://github.com/ksenkap/ant_alg